

2021.06.12. 교육

나정휘

<https://justicehui.github.io/>

목차

- 시간 복잡도
- 정렬
- 이분 탐색
- 매개변수 탐색(Parametric Search)
- 삼분 탐색
- 단조 스택(Monotone Stack)

시간 복잡도

알고리즘의 성능을 판단하는 척도

- **정확성:** 얼마나 **정확한 답**을 구할 수 있는가?
- **작업량:** 얼마나 **적은 연산**을 필요로 하는가?
- **메모리 사용량:** 얼마나 **적은 공간**을 사용하는가?
- **단순성:** 알고리즘의 **작동 과정**이 얼마나 **단순**한가?
- **최적성:** 더 이상 **개선할 여지가 없을 만큼** 최적화가 잘 되었나?

시간 복잡도

- 문제를 해결하는데 **걸리는 시간**과 **입력 크기**의 함수 관계
 - 연산이 많아질수록 오래 걸림
 - PS에서는 네트워크 통신, CPU 점유 등은 신경 안 씀
 - 연산이 몇 번 수행되는지 확인하면 대략적으로 유추할 수 있음
 - **최악의 경우**가 중요함
- 문제를 해결하는데 필수적인 **기본 연산**
- 기본 연산의 **수행 횟수**

시간 복잡도 - 예시

- 길이가 N인 배열에서 값이 K인 원소가 존재하는지 확인
 - 기본 연산: 배열 원소 하나와 K의 값을 비교
 - 기본 연산의 수행 횟수: 최대 N회
 - 시간 복잡도: $T(N) = N$

시간 복잡도 - 예시

- 길이 N인 배열에서 최댓값 찾기
 - 기본 연산: 배열의 두 원소의 값 비교
 - 기본 연산의 수행 횟수: 최대 N-1회
 - 시간 복잡도 $T(N) = N-1$

질문?

함수에서 가장 중요한 정보

- $f(x) = x^2 + 5x, g(x) = 27x$
 - $x = 1: f(x) = 6, g(x) = 27$
 - $x = 10: f(x) = 150, g(x) = 270$
 - $x = 100: f(x) = 10500, g(x) = 2700$
 - $x = 1000: f(x) = 1005000, g(x) = 27000$
- x 가 적당히 큰 수면 $f(x) > g(x)$ 를 만족함
- $x > X$ 이면 항상 $f(x) > g(x)$ 를 만족함

함수에서 가장 중요한 정보

- 다항 함수: 최고차항의 차수가 중요
 - ex) $f(x) = x^2 + 5x, g(x) = 27x$
 - $x > 220$ 이면 항상 $f(x) > g(x)$
- 지수 함수는 밑수의 최댓값이 중요
 - $f(x) = 3^x + x - 10, g(x) = 2^x + x^3 + 5x$
 - $x > 4.6$ 이면 항상 $f(x) > g(x)$
- 최고차항의 차수 or 밑수의 최댓값이 큰 함수가 더 크다.

Big-O Notation

- $f(x) \in O(g(x))$
 - 어떤 실수 x_0 과 양의 실수 c 가 있어서
 - $x > x_0$ 을 만족하는 모든 x 에 대해
 - $f(x) \leq c g(x)$ 를 만족한다.
- 어떤 실수 x_0 보다 큰 모든 x
 - x 가 한 없이 커지면 항상 부등호가 성립
- 어떤 양의 실수 c
 - 최고차항의 계수 무시
 - 최고차항의 차수와 지수 항의 밑수가 중요함

Big-O Notation

- $f(x) \in O(g(x))$
 - $f(x)$ 가 아무리 빨리 증가해도
 - **최대** $g(x)$ 에 비례하는 수준으로 증가한다
 - $f(x)$ 의 **상계**를 나타냄

Big-O Notation – 예시

- $f(x) = 5x, g(x) = x^2$
 - $x_0 = 5, c = 1$ 이면 $5x \leq 1 \times x^2 : 5x \in O(x^2)$
- $f(x) = 2^x + 10x + 5, g(x) = 2^x$
 - $x_0 = 7, c = 2$ 이면 $2^x + 10x + 5 \leq 2 \times 2^x : 2^x + 10x + 5 \in O(2^x)$

질문?

Big-Omega Notation

- Big-O Notation과 반대로 함수의 **하계**를 나타냄
- $f(x) \in \Omega(g(x))$
 - 어떤 실수 x_0 과 양의 실수 c 가 있어서
 - $x > x_0$ 을 만족하는 모든 x 에 대해
 - $f(x) \geq c g(x)$ 를 만족한다.

Big-Theta Notation

- 상계와 하계의 교집합(Tight Bound)
- $f(x) \in \Theta(g(x)) \iff f(x) \in O(g(x)) \wedge f(x) \in \Omega(g(x))$

극한을 이용한 표현

- $f(x) \in O(g(x)) \iff \lim_{n \rightarrow \infty} \frac{f(x)}{g(x)} = c$ 로 수렴
- $f(x) \in \Omega(g(x)) \iff \lim_{n \rightarrow \infty} \frac{f(x)}{g(x)} > 0$
- $f(x) \in \Theta(g(x)) \iff \lim_{n \rightarrow \infty} \frac{f(x)}{g(x)} = c$ 로 수렴 (단, $c > 0$)

질문?

더 공부할 거리

- Amortized Analysis (중요)
- Master Theorem

정렬

정렬이란?

- 주어진 데이터를 일정한 순서대로 나열하는 것
 - 오름차순 정렬: $a < b$ 이면 a 가 b 보다 앞에 오도록 나열
 - 내림차순 정렬: $a > b$ 이면 a 가 b 보다 앞에 오도록 나열
 - 위상 정렬: $a \rightarrow b$ 간선이 있으면 a 가 b 보다 앞에 오도록 나열

비교 함수

- `bool Compare(T a, T b)`
 - a가 b보다 **반드시** 앞에 나와야 하면 `true`
 - 그렇지 않으면 `false`
 - **Strict Weak Ordering**을 만족해야 함
 - $i < j \ \&\& \text{Compare}(A[j], A[i]) == \text{true}$ 인 i, j 가 존재하지 않으면 됨
 - $i < j \ \&\& \text{Compare}(A[i], A[j]) == \text{false}$ 면 $A[i]$ 와 $A[j]$ 를 교환
- 오름차순: `Compare(a, b): a < b` $a \leq b$ 아님
- 내림차순: `Compare(a, b): a > b` $a \geq b$ 아님

Strict Weak Ordering

- Strict Weak Ordering
 - 비반사성(irreflexivity)
 - 모든 x 에 대해 $R(x, x)$ 는 거짓
 - 비대칭성(asymmetry)
 - 모든 x, y 에 대해 $R(x, y)$ 이 참이면 $R(y, x)$ 는 거짓
 - 추이성(transitivity)
 - 모든 x, y, z 에 대해 $R(x, y), R(y, z)$ 가 참이면 $R(x, z)$ 는 참
 - 비비교성의 추이성(transitivity of incomparability)
 - 모든 x, y, z 에 대해 $R(x, y), R(y, x), R(y, z), R(z, y)$ 가 거짓이면 $R(x, z), R(z, x)$ 는 거짓

Strict Weak Ordering-비교성

- 비교성(comparability)
 - 비교를 할 수 있다
 - (정렬에서) 두 원소의 순서를 정할 수 있다
- 비비교성(incomparability)
 - 비교를 할 수 없다
 - (정렬에서) 두 원소의 순서를 정할 수 없다 = 동등하다
- ex) 오름차순 정렬
 - 값이 같은 두 원소는 순서를 정할 수 없음
 - 값이 같은 두 원소는 순서를 정할 수 있음

질문?

Strict Weak Ordering-비반사성

- 비반사성(irreflexivity)
 - 모든 x 에 대해 $R(x, x)$ 는 거짓
 - 값이 같은 두 원소는 비교가 불가능하다
 - $\text{Compare}(x, x)$ 는 두 x 중 반드시 먼저 와야 하는 것을 결정할 수 없음
- 오름차순의 비교 함수로 $a \leq b$ 를 사용할 수 없는 이유

Strict Weak Ordering-비대칭성

- 비대칭성(asymmetry)
 - 모든 x, y 에 대해 $R(x, y)$ 이 참이면 $R(y, x)$ 는 거짓
 - 두 개가 모두 참이면 두 원소의 순서를 정할 수 없음
 - $R(x, y), R(y, x)$ 모두 거짓이 되어야 함
- 사이클이 있는 그래프에서 위상 정렬이 불가능한 이유

Strict Weak Ordering-추이성

- 추이성(transitivity)
 - 모든 x, y, z 에 대해 $R(x, y), R(y, z)$ 가 참이면 $R(x, z)$ 는 참
 - 삼단 논법

Strict Weak Ordering-비비교성의 추이성

- 비비교성의 추이성(transitivity of incomparability)
 - 모든 x, y, z 에 대해 $R(x, y), R(y, x), R(y, z), R(z, y)$ 가 거짓이면 $R(x, z), R(z, x)$ 는 거짓
 - $x \sim y$ 가 동등하고(비교가 불가능하고), $y \sim z$ 가 동등하면 $x \sim z$ 도 동등해야 함

질문?

std::sort 사용법

- `sort(first, last)`: 시작 주소, 끝 주소
- `sort(first, last, comp)`: 시작 주소, 끝 주소, 비교 함수

Stable Sort

- Stable Sort / Unstable Sort
- 동등한(비교 불가능한) 원소들의 순서는 어떻게 결정할까?
 - Stable Sort: 입력 순서대로
 - Unstable Sort: 마음대로
- `std::sort`는 Unstable Sort
- Stable Sort를 사용하고 싶다면 `std::stable_sort` 사용
 - 사용법 동일함

질문?

더 공부할 거리

- Intro Sort (`std::sort`의 구현체, 퀵정렬 + 힙정렬 + 삽입정렬)
- Tim Sort (`std::stable_sort`의 구현체)

이진 탐색

업/다운 게임

- 철수는 1 이상 N 이하인 자연수 X 를 하나 선택한다.
- 영희는 철수가 생각한 X 를 맞춰야 한다.
- 영희가 어떤 수 Y 를 말하면, 철수는 아래와 같은 대답을 한다.
 - $X > Y$ 라면: Up
 - $X < Y$ 라면: Down
 - $X = Y$ 라면: Accept
- 영희의 질문 횟수를 **최소화** 시키자.

영희의 전략

- 정답은 $[1, N]$ 사이에 있다.
 - **중간 지점** $M_1 = \text{floor}((1+N)/2)$ 을 선택한다.
 - 정답이 M_1 보다 작다면 구간을 $[1, M_1-1]$ 로 조절한다.
 - 정답이 M_1 보다 크다면 구간을 $[M_1+1, N]$ 으로 조절한다.
 - $[S_2, E_2]$ 라고 하자
- 정답은 $[S_2, E_2]$ 사이에 있다.
 - **중간 지점** $M_2 = \text{floor}((S_2+E_2)/2)$ 를 선택한다.
 - 정답이 M_2 보다 작다면 구간을 $[S_2, M_2-1]$ 로 조절한다.
 - 정답이 M_2 보다 크다면 구간을 $[M_2+1, E_2]$ 로 조절한다.
- 반복한다.

영희의 전략 분석

- 정답이 존재할 수 있는 구간이 매번 절반으로 감소한다.
 - 최악의 경우에도 $\log_2 N + 1$ 번의 질문으로 답을 구할 수 있다.
- 이게 최적 전략일까?
 - YES
 - 증명이 궁금하면 상대자 논증을 검색해보자.

이분 탐색

- 정렬된 배열 A가 주어진다. K가 A의 몇 번째 원소인지 구해라.
- 이분 탐색으로 풀 수 있겠죠?
- 배열의 길이를 N이라고 하면 $O(\log N)$ 에 해결할 수 있다.
 - 기본 연산: 배열의 원소와 어떤 수를 비교하는 것
 - 기본 연산의 수행 횟수: 최악의 경우 $\log_2 N + 1$ 번

질문?

더 공부할 거리

- 하계 이론
- 상대자 논증
- 매개변수 탐색: 뒤에서 다룸
- 삼분 탐색: 뒤에서 다룸

매개변수 탐색

결정 문제와 최적화 문제

- 결정 문제: YES/NO로 답할 수 있는 문제
- 최적화 문제: 최댓값/최솟값을 구하는 문제
- 어떤 것이 더 어려울까?
 - 최적화 문제가 결정 문제보다 쉬울 수 없다.
 - 최적화 문제를 풀 수 있으면 무조건 결정 문제를 풀 수 있다.

결정 문제와 최적화 문제 – 예시

- 배열 A의 최댓값을 구해라 (최적화 문제)
- 배열 A의 최댓값이 3 이하인지 판단해라 (결정 문제)
- 최적화 문제를 푸는 함수 $\text{optimize}(A)$ 가 있으면
- 결정 문제를 푸는 함수 $\text{decision}(A) := \text{optimize}(A) \leq 3$

매개변수 탐색

- 최적화 문제를 결정 문제로 바꿔서 푸는 방법
 - 배열 A의 최댓값을 구하는 최적화 문제
 - 배열 A의 최댓값이 K 이하인지 판별하는 결정 문제
 - $K = 0, 1, 2, \dots$ 에 대해 결정 문제를 풀면 된다.
- 더 비효율적인데?

매개변수 탐색

- 어떠한 경우에 매개변수 탐색이 효율적일까?
 - 최댓값을 구할 때: 결정 문제의 답이 YYYYYY...YNNN...N인 경우
 - 최솟값을 구할 때: 결정 문제의 답이 NNNN...NYYYYY...Y인 경우
- 이분 탐색을 응용하면 풀 수 있다
 - 최댓값을 구할 때: 정답이 존재하는 구간 $[S, E]$ 와 중간 지점 M 에 대해
 - $\text{Decision}(M)$ 이 참이면 정답은 $[M, E]$ 에 존재
 - $\text{Decision}(M)$ 이 거짓이면 정답은 $[S, M-1]$ 에 존재
 - 최솟값을 구할 때: 정답이 존재하는 구간 $[S, E]$ 와 중간 지점 M 에 대해
 - $\text{Decision}(M)$ 이 참이면 정답은 $[S, M]$ 에 존재
 - $\text{Decision}(M)$ 이 거짓이면 정답은 $[M+1, E]$ 에 존재

매개변수 탐색 - 구현

```
int Minimization(){
    int s = 1, e = N;
    while(s < e){
        int m = (s + e) / 2; // floor((s + e) / 2)
        if(Decision(m)) e = m;
        else s = m + 1;
    }
    return e;
}

int Maximization(){
    int s = 1, e = N;
    while(s < e){
        int m = (s + e + 1) / 2; // ceil((s + e) / 2)
        if(Decision(m)) s = m;
        else e = m - 1;
    }
    return s;
}
```

정수 범위라면 이것도 가능하다!

- 첫 번째 단계에서 $m = 2^{(K-1)}$
- 두 번째 단계에서 $m = 2^{(K-2)}$
- i 번째 단계에서 $m = 2^{(K-i)}$

```
constexpr int K = 18;
int Maximize(){
    // int l = 0, r = (1 << K) - 1;
    int ans = 0;
    for(int bit=K-1; bit>=0; bit--){
        if(Decision(ans | 1 << bit)) ans |= 1 << bit;
    }
    return ans;
}
```


실수 범위 이분 탐색

- 충분히 많이 돌리는 것이 정신 건강에 이롭다.
 - 탐색해야 할 구간의 길이가 X 이고, 오차를 $\frac{1}{K}$ 까지 허용한다면
 - $\log_2(XK)$ 번 이상 해야 함 (\because 매번 가능한 오차의 범위가 2배 감소함)

```
using ld = long double;
ld Maximize(){
    ld l = 0, r = 1e5;
    for(int i=0; i<100; i++){
        ld m = (l + r) / 2;
        if(Decision(m)) l = m;
        else r = m;
    }
    return l;
}
```

질문?

삼분 탐색

유니모달 함수

- 함수 $f(x)$ 가 유니모달(unimodal)하다.
 - $x_1 < x_2$ 인 x_1 과 x_2 대해, 아래 두 식을 만족하는 x^* 가 존재한다.
 - $x_2 < x^*$ 이면 $f(x_1) > f(x_2)$
 - $x_1 > x^*$ 이면 $f(x_1) < f(x_2)$
- **최대가 되는 지점** x^* 를 기준으로
 - x^* 이전에서는 감소 함수
 - x^* 이후에서는 증가 함수

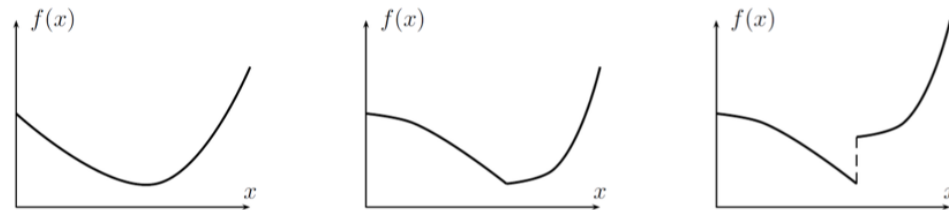


Figure 2. Examples of unimodal functions.

삼분 탐색

- 유니모달 함수가 최대가 되는 지점을 찾는 알고리즘
- 최댓값이 존재할 수 있는 구간을 삼등분, $m_1 \leq m_2$ 라고 하자.
 - $f(m_1) < f(m_2)$ 이면 최댓값은 $(m_1, e]$ 에 있음
 - $f(m_1) > f(m_2)$ 이면 최댓값은 $[s, m_2)$ 에 있음
- 매번 구간이 $\frac{2}{3}$ 가 되므로 $\log_{1.5} N$ 단계를 거치면 됨 : $O(\log N)$

삼분 탐색 - 구현

- 정수 범위: while 문 종료 조건 처리 귀찮음 → 아래 코드 참고
- 실수 범위: 충분히 많이 돌리자. ($\log_{1.5}(XK)$ 번 이상)

```
int FindMaximalPosition(int s, int e){
    while(e-s > 3){
        int m1 = (s + s + e) / 3;
        int m2 = (s + e + e) / 3;
        if(f(m1) < f(m2)) s = m1;
        else e = m2;
    }
    int pos = s;
    for(int i=s; i≤e; i++){
        if(f(pos) < f(i)) pos = i;
    }
    return pos;
}
```

질문?

더 공부할 거리

- 경사 하강법

단조 스택 (Monotone Stack)

BOJ 17298 오큰수

- 크기가 N 인 수열 A 가 주어진다.
- 오큰수 $NGE(i)$
 - A_i 보다 오른쪽에 있으면서 A_i 보다 큰 수 중 가장 왼쪽에 있는 수
- $NGE(1), NGE(2), \dots, NGE(N)$ 을 구하는 문제

Monotone Stack

- 스택의 원소를 순증가/순감소/단조증가/단조감소 상태로 유지
 - 순증가: $A[i] < A[i+1]$
 - 순감소: $A[i] > A[i+1]$
 - 단조증가: $A[i] \leq A[i+1]$
 - 단조감소: $A[i] \geq A[i+1]$
- 만약 순감소 상태를 유지한다면...
 - X를 스택에 넣기 전에
 - X보다 작거나 같은 원소를 모두 스택에서 제거하고
 - X를 스택에 넣는다

Monotone Stack – 예시

- 4 3 3 2 4 2 1을 차례대로 넣어보자.
 - 4 삽입 : $S = \{4\}$
 - 3 삽입 : $S = \{4, 3\}$
 - 3 삽입 (3 제거)
 - 3 제거 : $S = \{4\}$
 - 3 삽입 : $S = \{4, 3\}$
 - 2 삽입 : $S = \{4, 3, 2\}$
 - 4 삽입 (2, 3, 4 제거)
 - 2, 3, 4 제거 : $S = \{\}$
 - 4 삽입 : $S = \{4\}$
 - 2 삽입 : $S = \{4, 2\}$
 - 1 삽입 : $S = \{4, 2, 1\}$

Monotone Stack

- X를 삽입하기 직전에 스택의 맨 위에 있는 원소
 - X보다 앞에 있으면서 X보다 큰 원소 중 가장 나중에 나온 원소
- 4 삽입 : $S = \{4\}$
- 3 삽입 : $S = \{4, 3\}$
- 3 삽입 (3 제거)
 - 3 제거 : $S = \{4\}$
 - 3 삽입 : $S = \{4, 3\}$
- 2 삽입 : $S = \{4, 3, 2\}$
- 4 삽입 (2, 3, 4 제거)
 - 2, 3, 4 제거 : $S = \{\}$
 - 4 삽입 : $S = \{4\}$
- 2 삽입 : $S = \{4, 2\}$
- 1 삽입 : $S = \{4, 2, 1\}$

Monotone Stack

- 순증가: X 앞에 있으면서 X 미만 중 가장 나중에 나온 원소
 - 순감소: X 앞에 있으면서 X 초과 중 가장 나중에 나온 원소
 - 단조증가: X 앞에 있으면서 X 이하 중 가장 나중에 나온 원소
 - 단조감소: X 앞에 있으면서 X 이상 중 가장 나중에 나온 원소
-
- 스택이 비어있다: 그런 원소 없다
 - `stk.empty()` 검사하기 귀찮으면 초기화할 때 INF 값 넣어주면 됨

BOJ 17298 오큰수

- A_i 보다 뒤에 있으면서 A_i 초과 중 가장 먼저 나온 원소
- 배열 A 를 뒤집고, 스택을 순감소 상태로 유지하면 됨

질문?

과제

Div.2

5	A - 수 정렬하기 2
5	B - 좌표 정렬하기
4	C - 국영수
1	D - 2차원 배열의 합
4	E - 수 찾기
3	F - 나무 자르기
3	G - 예산
2	H - 합이 0인 네 정수
5	I - 날카로운 눈
4	J - 삶의 질

Div.1

2	H - 합이 0인 네 정수
5	I - 날카로운 눈
4	J - 삶의 질
4	K - 오큰수
5	L - 히스토그램에서 가장 큰 직사각형
4	M - 수열의 값
5	N - 전봇대
5	O - 먼 별
4	P - 조화로운 행렬
3	Q - Aliens